Treasure Cave - Development Update #0

Blog - Programming - 04

Luca Hohmann
business@lucahohmann.com

11. May 2019

Abstract

In today's blog, we're going to take a look into the development of Treasure Cave over the course of the last 2.5 weeks. It is mainly focused around the GUI which is has the main focus at the moment, as well as UI animations in Unity3D.

Figure 0.1 Treasure Cave logo (will get a redesign as well)

1 Introduction

In the last two weeks a few important actions took place. The most important news coming from these is, that we are only developing the game as 2 people. The reason behind this situation is, that the other ones left the project for their own personal reasons. As a result of this, Maarten and I sat down, and agreed on finishing the game in the state it is right now. So we are going to finish the GUI, implement the new level and make some balancing changes.

Because the GUI has to be finished before we can move on to other parts, I'm focusing on this part as much as possible. When we worked on the project during the course of the lecture "Interaction methods and Devices", I had already created a 3D menu scene, which did not find it's way into the game yet. But now with the GUI overhaul, I'm creating the GUI for this exact menu background (Figure 1.1).

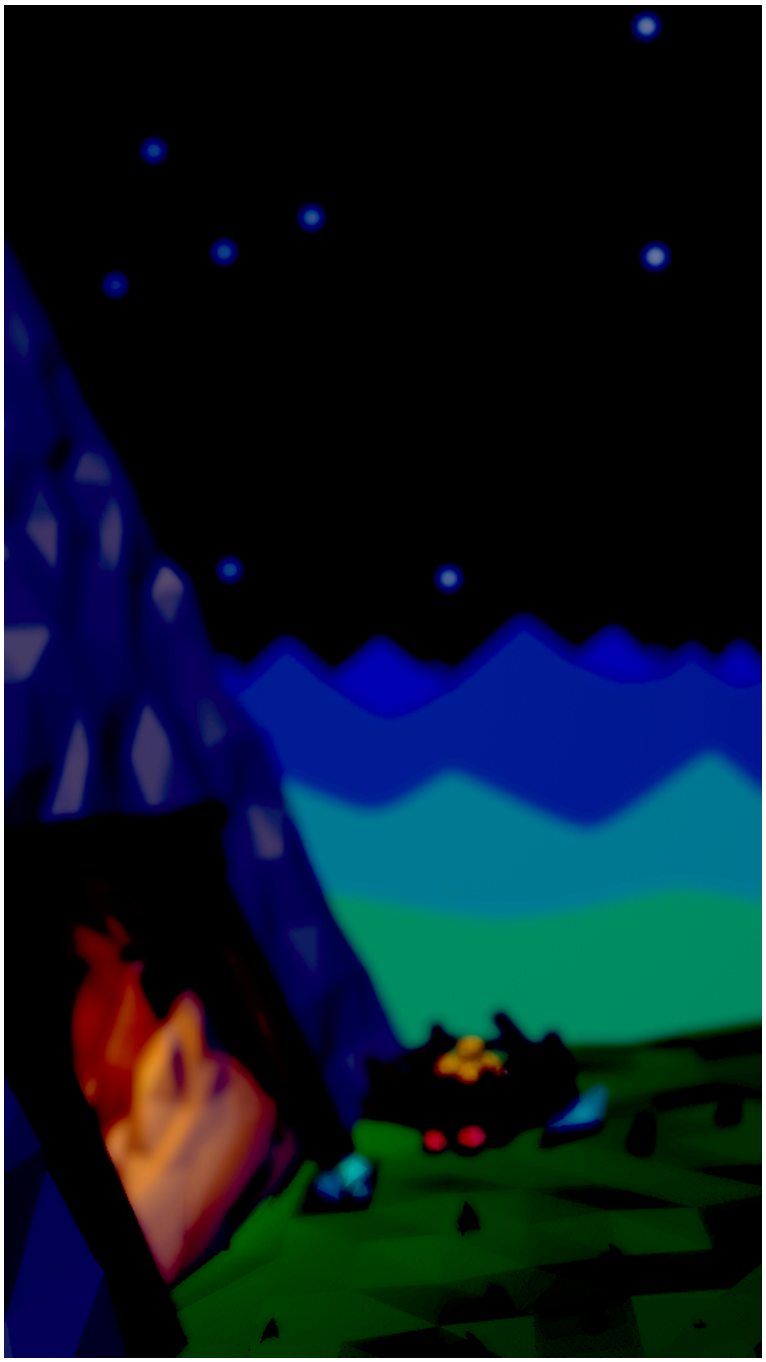

Figure 1.1 background of the new main menu

2 The new GUI

2.1 The UI elements

The first thing for a GUI overhaul is agreeing on a new style for the UI elements. I already had something in mind and after some testing, it turned out to be the best solution for us. Of course, there might be much more beautiful and artistic solutions, but none of us is a 2D graphic designer, so we went for the simple yet fitting design seen in Figure 2.1.

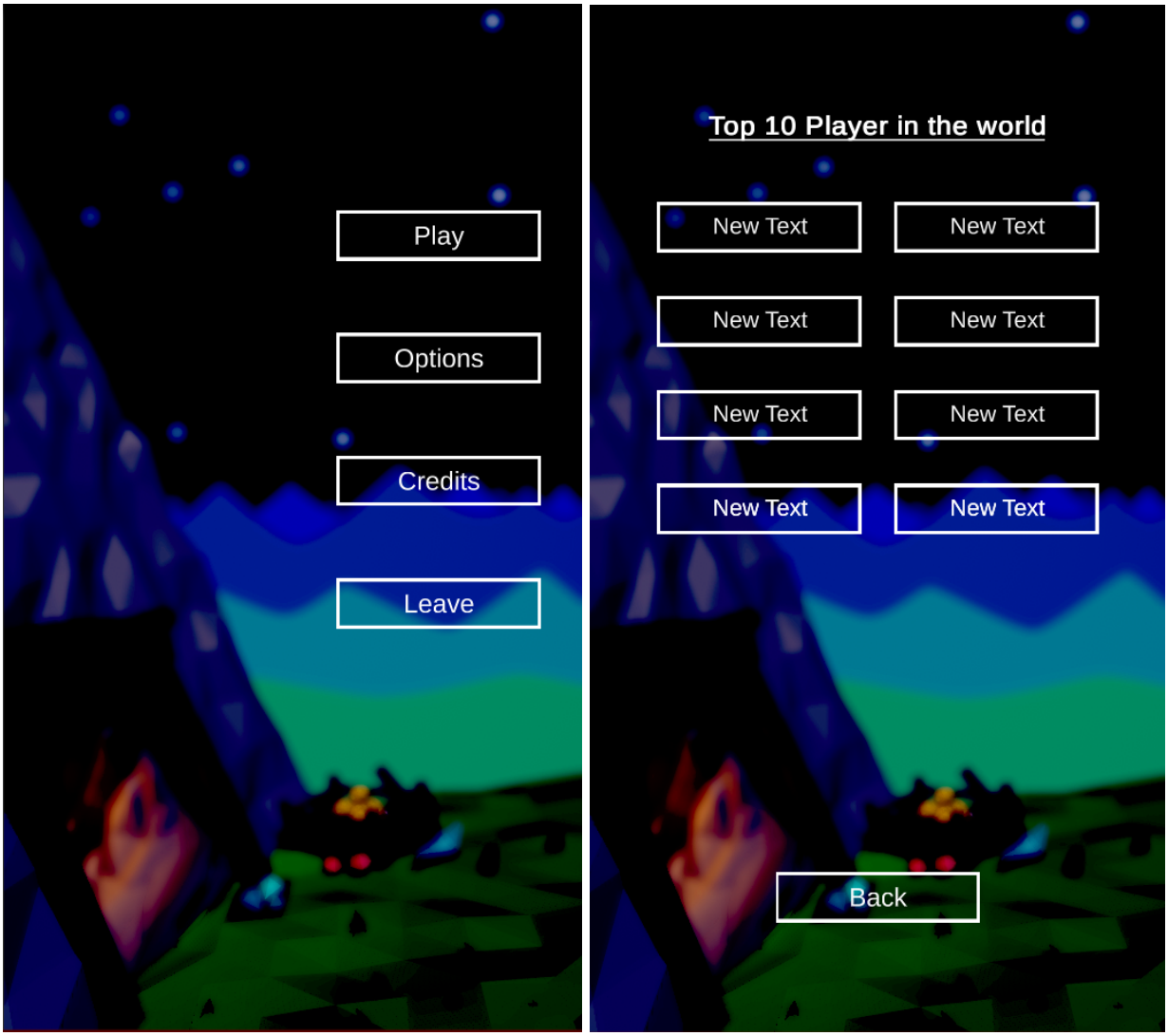

Figure 2.1 UI style in 2 different menus

It is always clear what every text says, and the white color is a great contrast to the background. Another theoretical advantage of this design is, that even with changing background colors, the white is still clearly visible or can be changed easily through the corresponding material.

Because the UI elements are very basic, I only want to point out one special feature in the play menu. The game will have a level based playmode, so the player should be able to choose between levels. In order to give the player a better understanding of the differences in each level, the level menu displays the 3D model in a rotating motion. As seen in Figure 2.2, the inside of the levels are always visible.

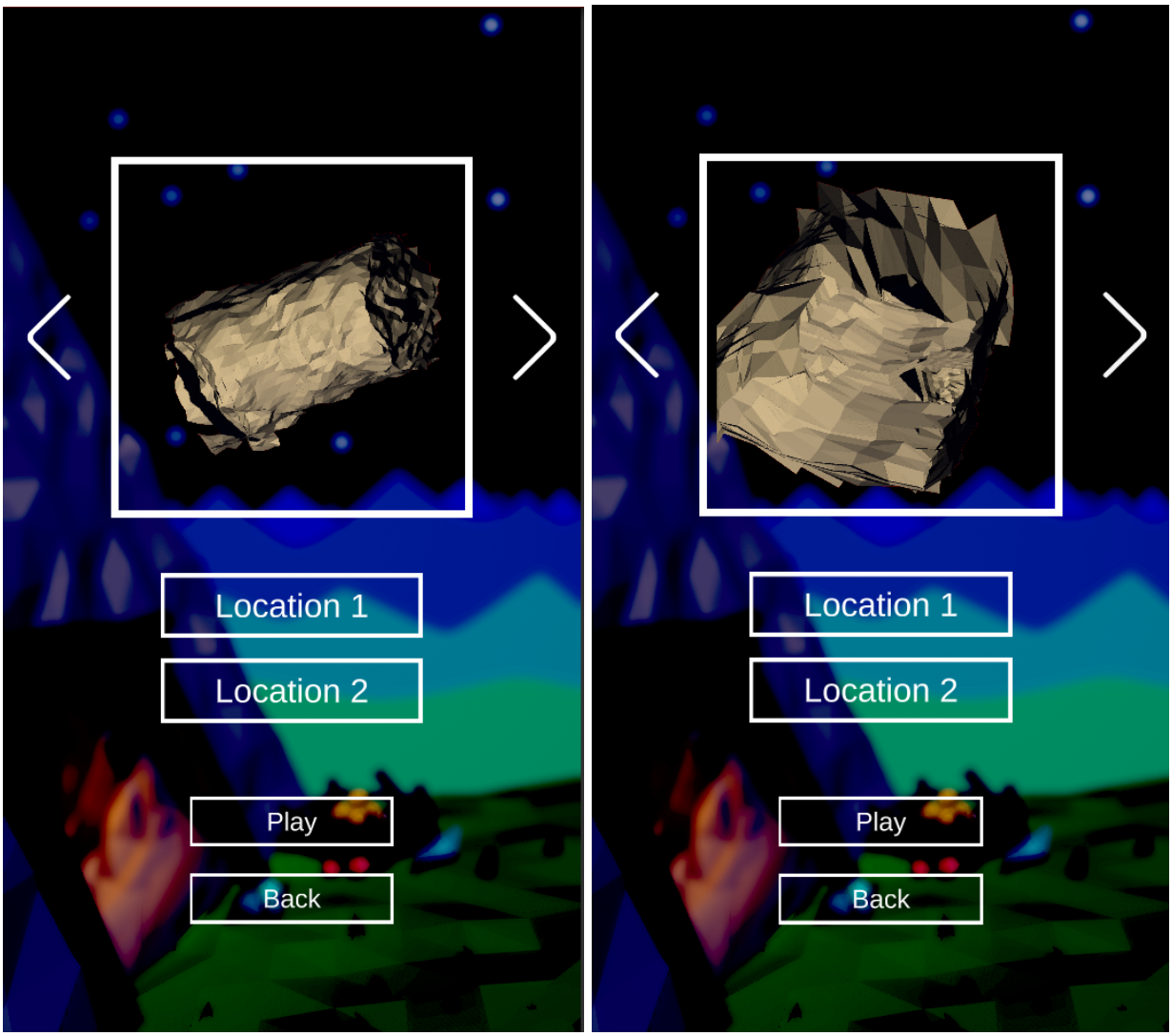

Figure 2.2 Level menu screen

This effect is very easy to achieve in this case. Because all normal vectors face into one direction (inside the tube), Unity does not render the faces which would otherwise block the view.

2.2 GUI animations

A very important part of ever GUI system are animations. Without these, applications do not feel very interactive, mainly because then the UI elements just disappear and new ones appear in a blink of an eye. This is usually considered as too fast and instant, which affects the user experience in a negative way. Of course it is important not to create the opposite of an instant UI. A GUI which has very slow animations does not feel interactive as well. As a result, I decided to implement simple animations, which affect only 2 parameters of each element, and therefore are easy to control and change. Every UI element fades and moves in. So only the alpha value as well as the position on one axis (either x or y) is manipulated for each animation. Besides the easy control, this offers a big advantage towards reusability. Every fade in animation can be used as fade out animation, simply by setting the playback rate to -1.

Every animation sequence uses the same animator state machine seen in Figure 2.3. It starts with an entry waiting state, which is used until the UI element appears the first time. From there on, the FadeIn and FadeOut state contain the animations, while the wait states only have an empty animation. The wait states are used to delay the the animation until the previous UI has faded out. Every time an UI element is touched and new UI elements have to be loaded, the UI manager script fires an event to the corresponding trigger variable for the affected UI elements. The advantage of trigger variables as conditions in the state machine is the automatic reset after the condition has be activated. This

eliminates the necessity to implement such a system with coroutines or the use of the Update method. The UI manager works on arrays of animators, which are assigned via the editor drag n drop system. This way, one method for event handling can be reused by simply introducing a parameter each UI element has to provide when activated. With this index the corresponding group of UI elements can be assigned to the working array in the function. This is also possible because the two triggers in all animators are named the same way, so it is always possible to trigger this condition, regardless of the actual animator.

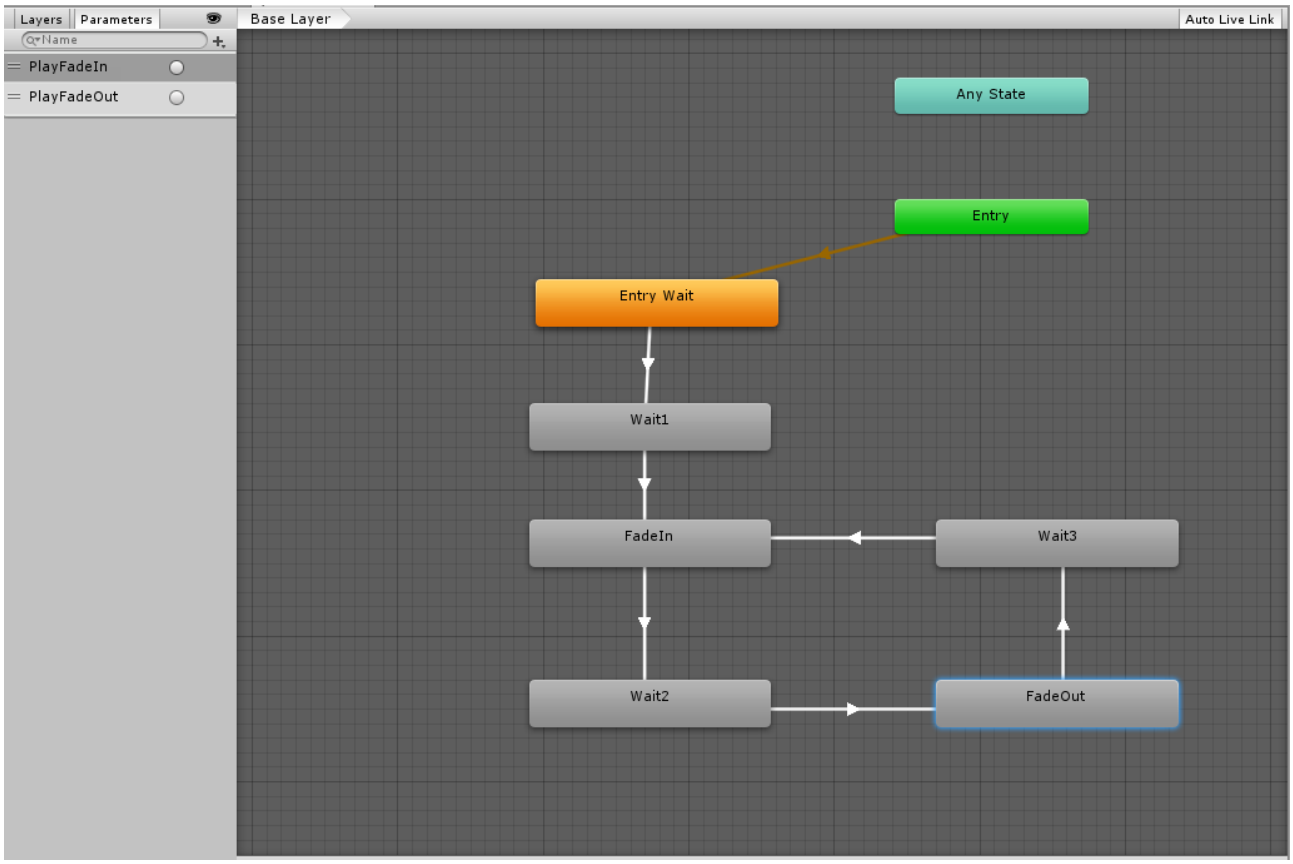

Figure 2.3 Base animator state machine for all UI animations

3 Final words

Because not much happened besides me working on the UI, I'm gonna end this blog here. I hope to finish the UI within a few weeks, simply because university has started again and so a lot of additional work is ahead.

So if you want to stay updated on the game, follow me on Twitter: @lucahohmann.
And yes, thanks for reading, have a good day and see you next time
Luca =)